Bioss-IA 2020 Workshop

GULA: Learning (From Any) Semantics of a Biological Regulatory Network

Maxime FOLSCHETTE · http://maxime.folschette.name/
Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

Tony RIBEIRO · http://www.tonyribeiro.fr/
Independent Researcher +
Laboratoire des Sciences du Numérique de Nantes, 44321 Nantes, France +
National Institute of Informatics, Tokyo 101-8430, Japan

Joint work with Morgan MAGNIN (ECN + LS2N + NII)
and Katsumi INOUE (NII + SOKENDAI + Tokyo Tech)

2020-11-24

Introduction

Learn interaction rules from the dynamical transitions

- LFIT: **synchronous** semantics, deterministic (Boolean)
  [Inoue, Ribeiro, Sakama, *Machine Learning Jour.*, 2014]
- LFkT: **synchronous** semantics, with memory (Boolean)
  [Ribeiro, Magnin, Inoue, Sakama, *Frontiers in Bioeng. and Biotech.*, 2015]
- LUST: **synchronous** semantics, non-deterministic
  [Martinez, Ribeiro, Inoue, Alenya, Torras, *ICLP*, 2015.]
- ACEDIA: **synchronous** semantics, continuous domains
  [Ribeiro, Tourret, Folschette, +5, *ILP*, 2017]
- **GULA**: **synchronous, asynchronous, general** semantics
  [Ribeiro, Folschette, Magnin, Roux, Inoue, *ILP*, 2018]

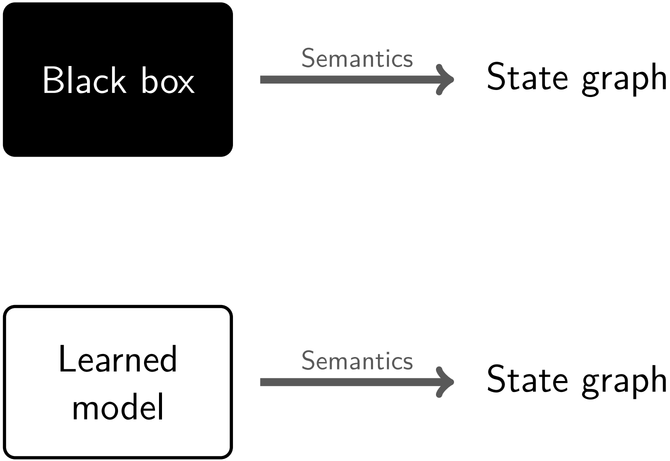
Content of this presentation: improvements on GULA

→ Define the scope of "learnable" semantics

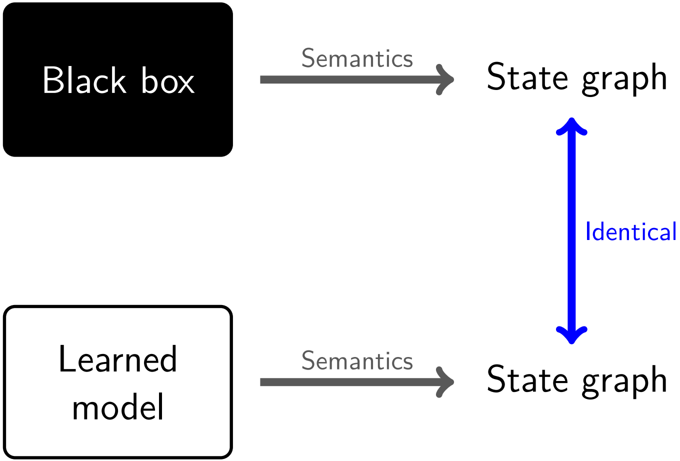
→ Learn the rules of the semantics itself

→ ...and more!

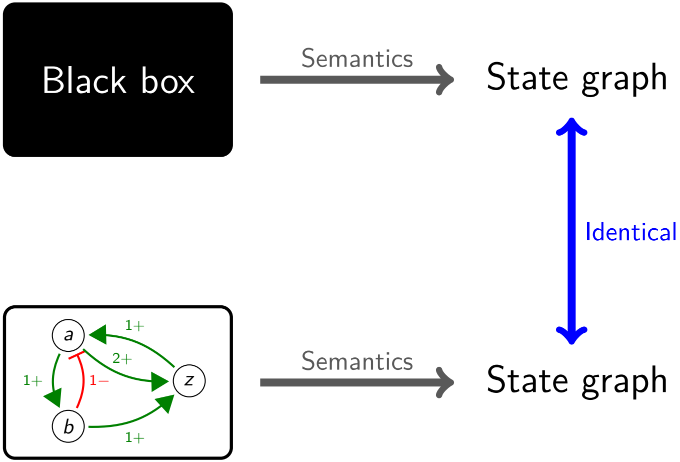
Introduction



Introduction

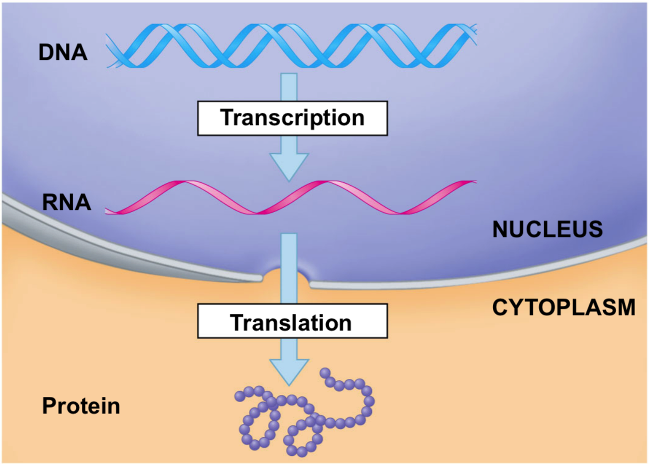


Introduction



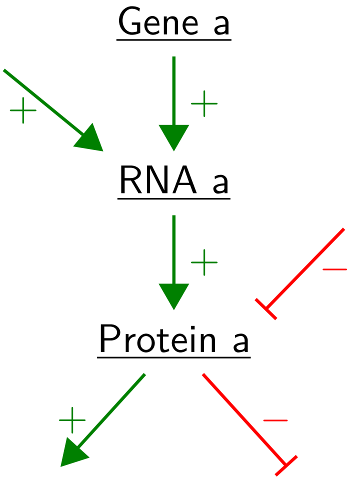
Discrete Networks

Preliminary Abstraction

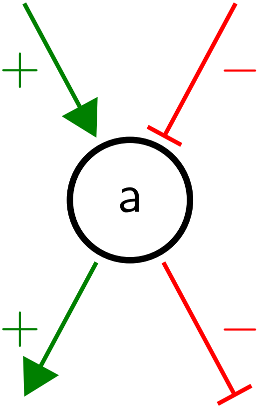


© 2012 Pearson Education, Inc.

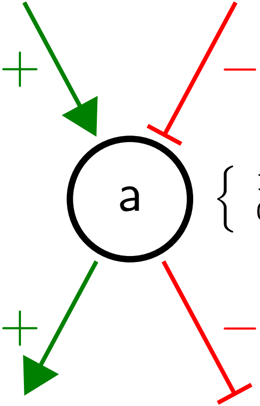
Preliminary Abstraction



Preliminary Abstraction

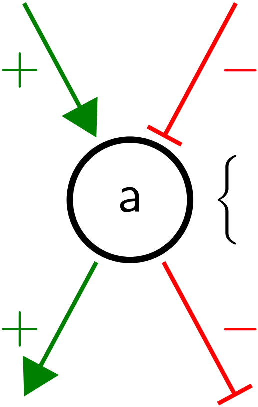


Preliminary Abstraction



$$\left\{ \begin{array}{l} 1 = \text{active} \\ 0 = \text{inactive} \end{array} \right\}$$

Preliminary Abstraction

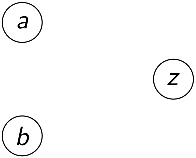


$$\left\{ \begin{array}{l} 2 = \text{saturation} \\ 1 = \text{traces} \\ 0 = \text{complete degradation} \end{array} \right\}$$

Discrete Networks / Thomas Modeling

[Kauffman, *Journal of Theoretical Biology*, 1969]
[Thomas, *Journal of Theoretical Biology*, 1973]

- A set of components $N = \{a, b, z\}$
- A discrete domain for each component $\mathrm{dom}(a) = [\![0;2]\!]$
- Discrete parameters / evolution functions $f^a : \mathcal{S} \to \mathrm{dom}(a)$
- Signs & thresholds on the edges (redundant) $a \xrightarrow{2+} z$



a	f^b
0	**0**
1	**1**
2	**1**

z	b	f^a
0	0	**1**
0	1	**0**
1	0	**1**
1	1	**2**

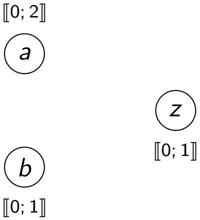
a	b	f^z
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**0**
2	0	**0**
2	1	**1**

Semantics = From this information, what are the next possible state(s)?

Discrete Networks / Thomas Modeling

[Kauffman, *Journal of Theoretical Biology*, 1969]
[Thomas, *Journal of Theoretical Biology*, 1973]

- A set of components $N = \{a, b, z\}$
- A discrete domain for each component $\mathrm{dom}(a) = [\![0; 2]\!]$
- Discrete parameters / evolution functions $f^a : \mathcal{S} \to \mathrm{dom}(a)$
- Signs & thresholds on the edges (redundant) $a \xrightarrow{2+} z$



a	f^b
0	**0**
1	**1**
2	**1**

z	b	f^a
0	0	**1**
0	1	**0**
1	0	**1**
1	1	**2**

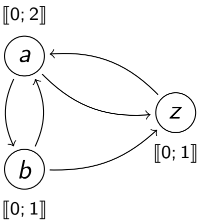
a	b	f^z
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**0**
2	0	**0**
2	1	**1**

Semantics = From this information, what are the next possible state(s)?

Discrete Networks / Thomas Modeling

[Kauffman, *Journal of Theoretical Biology*, 1969]
[Thomas, *Journal of Theoretical Biology*, 1973]

- A set of components $N = \{a, b, z\}$
- A discrete domain for each component $\mathrm{dom}(a) = [\![0; 2]\!]$
- Discrete parameters / evolution functions $f^a : \mathcal{S} \to \mathrm{dom}(a)$
- Signs & thresholds on the edges (redundant) $a \xrightarrow{2+} z$



a	f^b
0	**0**
1	**1**
2	**1**

z	b	f^a
0	0	**1**
0	1	**0**
1	0	**1**
1	1	**2**

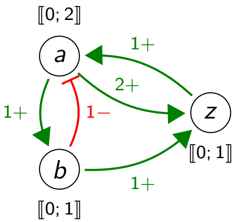
a	b	f^z
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**0**
2	0	**0**
2	1	**1**

Semantics = From this information, what are the next possible state(s)?

Discrete Networks / Thomas Modeling

[Kauffman, *Journal of Theoretical Biology*, 1969]
[Thomas, *Journal of Theoretical Biology*, 1973]

- A set of components $N = \{a, b, z\}$
- A discrete domain for each component $\mathrm{dom}(a) = [\![0; 2]\!]$
- Discrete parameters / evolution functions $f^a : \mathcal{S} \to \mathrm{dom}(a)$
- Signs & thresholds on the edges (redundant) $a \xrightarrow{2+} z$



a	f^b
0	**0**
1	**1**
2	**1**

z	b	f^a
0	0	**1**
0	1	**0**
1	0	**1**
1	1	**2**

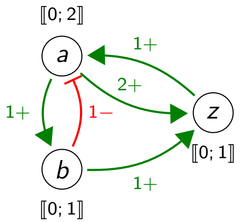
a	b	f^z
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**0**
2	0	**0**
2	1	**1**

Semantics = From this information, what are the next possible state(s)?

Discrete Networks / Thomas Modeling

[Kauffman, *Journal of Theoretical Biology*, 1969]
[Thomas, *Journal of Theoretical Biology*, 1973]

- A set of components $N = \{a, b, z\}$
- A discrete domain for each component $\mathrm{dom}(a) = [\![0; 2]\!]$
- Discrete parameters / evolution functions $f^a : \mathcal{S} \to \mathrm{dom}(a)$
- Signs & thresholds on the edges (redundant) $a \xrightarrow{2+} z$



a	f^b
0	**0**
1	**1**
2	**1**

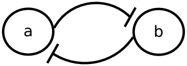
z	b	f^a
0	0	**1**
0	1	**0**
1	0	**1**
1	1	**2**

a	b	f^z
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**0**
2	0	**0**
2	1	**1**

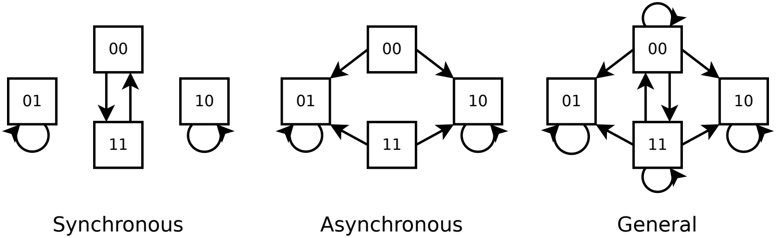
Semantics = From this information, what are the next possible state(s)?

Semantics

State transitions differ according to the update semantics used



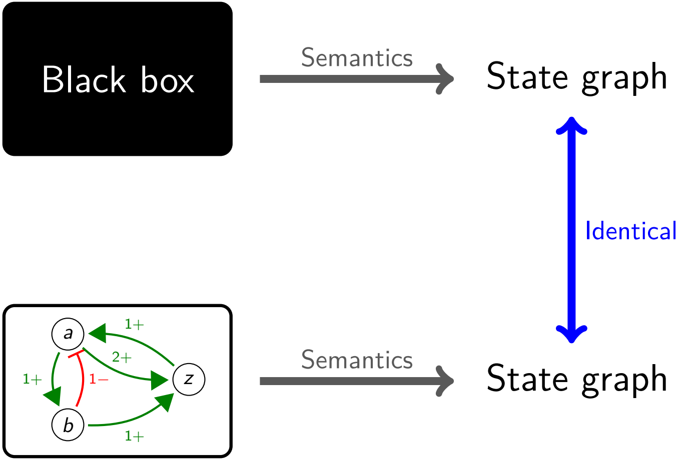
f(a) := not b.
f(b) := not a.



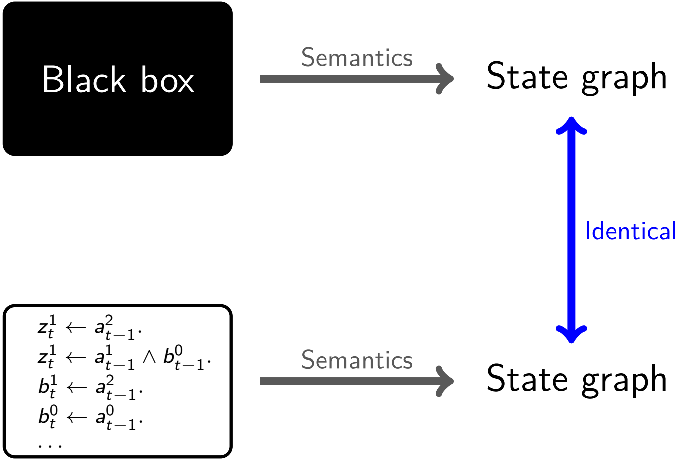
- **Synchronous**: all variables are updated
- **Asynchronous**: only one variable is updated
- **General**: any number of variables can be updated

Logic Programs

Principle of the Learning



Principle of the Learning



Logic Rule

$$\underbrace{\mathrm{v}_0^{val_0}}_{\substack{head \\ \text{target atom}}} \leftarrow \underbrace{\mathrm{v}_1^{val_1} \wedge \mathrm{v}_2^{val_2} \wedge \ldots \wedge \mathrm{v}_n^{val_n}}_{\substack{body \\ \text{feature atoms}}}.$$

- $\mathrm{v}_0, \mathrm{v}_1, \mathrm{v}_2, \ldots, \mathrm{v}_n$: variables $a_t, a_{t-1}, b_t, b_{t-1}, z_t, z_{t-1}$
  - Variables are split into feature ($\mathcal{F}$) and target ($\mathcal{T}$) variables
  - $\mathrm{v}_0 \in \mathcal{T}$ a_t, b_t, z_t
  - $\mathrm{v}_1, \mathrm{v}_2, \ldots, \mathrm{v}_n \in \mathcal{F}$ $a_{t-1}, b_{t-1}, z_{t-1}$
  - Implicit time step: t in the *head* and $t-1$ in the *body*

- $val_0, val_1, val_2, \ldots, val_n$: values 0, 1, 2, ...
  - $val_i \in \mathrm{dom}(\mathrm{v}_i)$

- All atoms in the *body* are in conjunction
- $\leftarrow$ is the (reverse) implication

Logic Rule

$$\underbrace{\mathrm{v}_0^{val_0}}_{\substack{\textit{head}\\ \text{target atom}}} \leftarrow \underbrace{\mathrm{v}_1^{val_1} \wedge \mathrm{v}_2^{val_2} \wedge \ldots \wedge \mathrm{v}_n^{val_n}}_{\substack{\textit{body}\\ \text{feature atoms}}}.$$

- $\mathrm{v}_0, \mathrm{v}_1, \mathrm{v}_2, \ldots, \mathrm{v}_n$: variables $a_t, a_{t-1}, b_t, b_{t-1}, z_t, z_{t-1}$
  - Variables are split into feature ($\mathcal{F}$) and target ($\mathcal{T}$) variables
  - $\mathrm{v}_0 \in \mathcal{T}$ a_t, b_t, z_t
  - $\mathrm{v}_1, \mathrm{v}_2, \ldots, \mathrm{v}_n \in \mathcal{F}$ $a_{t-1}, b_{t-1}, z_{t-1}$
  - Implicit time step: t in the *head* and $t-1$ in the *body*

- $val_0, val_1, val_2, \ldots, val_n$: values $0, 1, 2, \ldots$
  - $val_i \in \mathrm{dom}(\mathrm{v}_i)$

- All atoms in the *body* are in conjunction

- $\leftarrow$ is the (reverse) implication

Logic Rule

$$\underbrace{\mathrm{v}_0^{val_0}}_{\substack{head\\ \text{target atom}}} \leftarrow \underbrace{\mathrm{v}_1^{val_1} \wedge \mathrm{v}_2^{val_2} \wedge \ldots \wedge \mathrm{v}_n^{val_n}}_{\substack{body\\ \text{feature atoms}}}.$$

- $\mathrm{v}_0, \mathrm{v}_1, \mathrm{v}_2, \ldots, \mathrm{v}_n$: variables $a_t,\ a_{t-1},\ b_t,\ b_{t-1},\ z_t,\ z_{t-1}$
  - Variables are split into feature ($\mathcal{F}$) and target ($\mathcal{T}$) variables
  - $\mathrm{v}_0 \in \mathcal{T}$ $a_t,\ b_t,\ z_t$
  - $\mathrm{v}_1, \mathrm{v}_2, \ldots, \mathrm{v}_n \in \mathcal{F}$ $a_{t-1},\ b_{t-1},\ z_{t-1}$
  - Implicit time step: t in the *head* and $t-1$ in the *body*

- $val_0, val_1, val_2, \ldots, val_n$: values $0, 1, 2, \ldots$
  - $val_i \in \mathrm{dom}(\mathrm{v}_i)$

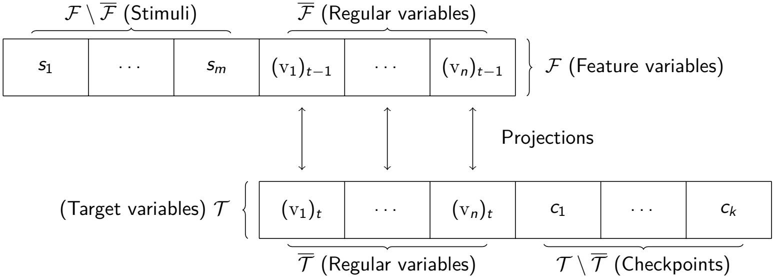
- All atoms in the *body* are in conjunction
- $\leftarrow$ is the (reverse) implication

Logic Rule

$$\underbrace{\mathrm{v}_0^{val_0}}_{\substack{\textit{head}\\ \text{target atom}}} \leftarrow \underbrace{\mathrm{v}_1^{val_1} \wedge \mathrm{v}_2^{val_2} \wedge \ldots \wedge \mathrm{v}_n^{val_n}}_{\substack{\textit{body}\\ \text{feature atoms}}}.$$

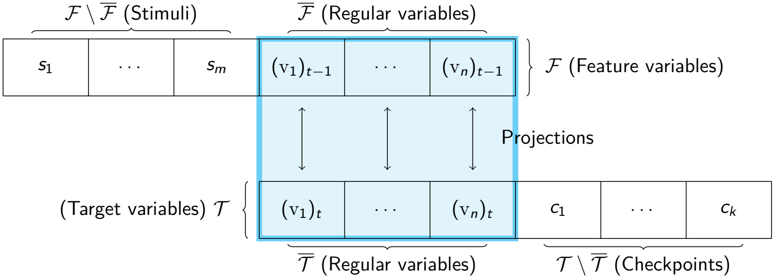
- $\mathrm{v}_0, \mathrm{v}_1, \mathrm{v}_2, \ldots, \mathrm{v}_n$: variables $a_t, a_{t-1}, b_t, b_{t-1}, z_t, z_{t-1}$
  - Variables are split into feature ($\mathcal{F}$) and target ($\mathcal{T}$) variables
  - $\mathrm{v}_0 \in \mathcal{T}$ a_t, b_t, z_t
  - $\mathrm{v}_1, \mathrm{v}_2, \ldots, \mathrm{v}_n \in \mathcal{F}$ $a_{t-1}, b_{t-1}, z_{t-1}$
  - Implicit time step: t in the *head* and $t-1$ in the *body*
- val_0, val_1, val_2, …, val_n: values 0, 1, 2, …
  - $val_i \in \mathrm{dom}(\mathrm{v}_i)$
- All atoms in the *body* are in conjunction
- $\leftarrow$ is the (reverse) implication

Feature & Target Variables



- **Feature variables** = causes
- Stimuli = known inputs
- **Target variables** = consequences
- Checkpoints = known outputs

Feature & Target Variables



- **Feature variables** = causes
- Stimuli = known inputs
- **Target variables** = consequences
- Checkpoints = known outputs

Interpretation of a Logic Rule

$$\underbrace{\mathrm{v}_0^{val_0}}_{\substack{\textit{head} \\ \text{target atom}}} \leftarrow \underbrace{\mathrm{v}_1^{val_1} \wedge \mathrm{v}_2^{val_2} \wedge \dots \wedge \mathrm{v}_n^{val_n}}_{\substack{\textit{body} \\ \text{feature atoms}}}.$$

Interpretation: When *body* is true, *head* is a potential outcome

Examples:

$$\left.\begin{array}{l} a_t^1 \leftarrow a_{t-1}^2 \wedge b_{t-1}^0 \wedge z_{t-1}^1. \\ b_t^1 \leftarrow z_{t-1}^1. \\ z_t^0 \leftarrow \top. \end{array}\right\} \text{all } \textbf{match } \langle a_{t-1}^2, b_{t-1}^0, z_{t-1}^1 \rangle$$

A rule R **matches** a state s iff *body* $\subseteq s$

Interpretation: When a state **matches** a rule,
the rule's *head* becomes a **candidate** for the next state

Semantics = From this information, what are the next possible state(s)?
(Similar to discrete networks)

Interpretation of a Logic Rule

$$\underbrace{\mathrm{v}_0^{val_0}}_{\substack{\textit{head}\\ \text{target atom}}} \leftarrow \underbrace{\mathrm{v}_1^{val_1} \land \mathrm{v}_2^{val_2} \land \dots \land \mathrm{v}_n^{val_n}}_{\substack{\textit{body}\\ \text{feature atoms}}}.$$

Interpretation: When *body* is true, *head* is a potential outcome

Examples:

$$\left.\begin{array}{l} a_t^1 \leftarrow a_{t-1}^2 \land b_{t-1}^0 \land z_{t-1}^1. \\ b_t^1 \leftarrow z_{t-1}^1. \\ z_t^0 \leftarrow \top. \end{array}\right\} \text{all } \textbf{match } \langle a_{t-1}^2, b_{t-1}^0, z_{t-1}^1 \rangle$$

A rule R **matches** a state s iff *body* $\subseteq s$

Interpretation: When a state **matches** a rule,
the rule's *head* becomes a **candidate** for the next state

Semantics = From this information, what are the next possible state(s)?
(Similar to discrete networks)

Interpretation of a Logic Rule

$$\underbrace{\mathrm{v}_0^{val_0}}_{\substack{\textit{head}\\ \text{target atom}}} \leftarrow \underbrace{\mathrm{v}_1^{val_1} \wedge \mathrm{v}_2^{val_2} \wedge \ldots \wedge \mathrm{v}_n^{val_n}}_{\substack{\textit{body}\\ \text{feature atoms}}}.$$

Interpretation: When *body* is true, *head* is a potential outcome

Examples:

$$\left.\begin{array}{l} a_t^1 \leftarrow a_{t-1}^2 \wedge b_{t-1}^0 \wedge z_{t-1}^1. \\ b_t^1 \leftarrow z_{t-1}^1. \\ z_t^0 \leftarrow \top. \end{array}\right\} \text{all } \textbf{match } \langle a_{t-1}^2, b_{t-1}^0, z_{t-1}^1 \rangle$$

A rule R **matches** a state s iff $body \subseteq s$

Interpretation: When a state **matches** a rule,
the rule's *head* becomes a **candidate** for the next state

Semantics = From this information, what are the next possible state(s)?
(Similar to discrete networks)

Interpretation of a Logic Rule

$$\underbrace{\mathrm{v}_0^{val_0}}_{\substack{\textit{head}\\ \text{target atom}}} \leftarrow \underbrace{\mathrm{v}_1^{val_1} \wedge \mathrm{v}_2^{val_2} \wedge \dots \wedge \mathrm{v}_n^{val_n}}_{\substack{\textit{body}\\ \text{feature atoms}}}.$$

Interpretation: When *body* is true, *head* is a potential outcome

Examples:
$$\left.\begin{array}{l} a_t^1 \leftarrow a_{t-1}^2 \wedge b_{t-1}^0 \wedge z_{t-1}^1. \\ b_t^1 \leftarrow z_{t-1}^1. \\ z_t^0 \leftarrow \top. \end{array}\right\} \text{ all } \textbf{match } \langle a_{t-1}^2, b_{t-1}^0, z_{t-1}^1 \rangle$$

A rule R **matches** a state s iff *body* $\subseteq s$

Interpretation: When a state **matches** a rule,
the rule's *head* becomes a **candidate** for the next state

Semantics = From this information, what are the next possible state(s)?
(Similar to discrete networks)

Interpretation of a Logic Rule

$$\underbrace{\mathrm{v}_0^{val_0}}_{\substack{\textit{head}\\ \text{target atom}}} \leftarrow \underbrace{\mathrm{v}_1^{val_1} \wedge \mathrm{v}_2^{val_2} \wedge \dots \wedge \mathrm{v}_n^{val_n}}_{\substack{\textit{body}\\ \text{feature atoms}}}.$$

Interpretation: When *body* is true, *head* is a potential outcome

Examples:

$$\left.\begin{array}{l} a_t^1 \leftarrow a_{t-1}^2 \wedge b_{t-1}^0 \wedge z_{t-1}^1. \\ b_t^1 \leftarrow z_{t-1}^1. \\ z_t^0 \leftarrow \top. \end{array}\right\} \text{ all } \textbf{match } \langle a_{t-1}^2, b_{t-1}^0, z_{t-1}^1 \rangle$$

A rule R **matches** a state s iff *body* $\subseteq s$

Interpretation: When a state **matches** a rule,
the rule's *head* becomes a **candidate** for the next state

Semantics = From this information, what are the next possible state(s)?
(Similar to discrete networks)

Interpretation of a Logic Rule

$$\underbrace{\mathrm{v}_0^{val_0}}_{\substack{\textit{head}\\ \text{target atom}}} \leftarrow \underbrace{\mathrm{v}_1^{val_1} \;\wedge\; \mathrm{v}_2^{val_2} \;\wedge\; \dots \;\wedge\; \mathrm{v}_n^{val_n}}_{\substack{\textit{body}\\ \text{feature atoms}}}.$$

Interpretation: When *body* is true, *head* is a potential outcome

Examples: $\left.\begin{array}{l} a_t^1 \leftarrow a_{t-1}^2 \wedge b_{t-1}^0 \wedge z_{t-1}^1. \\ b_t^1 \leftarrow z_{t-1}^1. \\ z_t^0 \leftarrow \top. \end{array}\right\}$ all **match** $\langle a_{t-1}^2, b_{t-1}^0, z_{t-1}^1 \rangle$

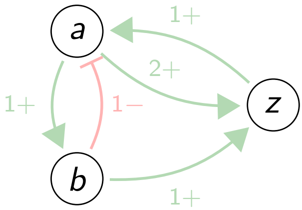
A rule R **matches** a state s iff *body* $\subseteq s$

Interpretation: When a state **matches** a rule,
the rule's *head* becomes a **candidate** for the next state

Semantics = From this information, what are the next possible state(s)?
(Similar to discrete networks)

Discrete Model as a Logic Program

Discrete model:



+ Discrete parameters
or evolution functions

Logic program:

$b_t^1 \leftarrow a_{t-1}^1.$

$b_t^1 \leftarrow a_{t-1}^2.$

$b_t^0 \leftarrow a_{t-1}^0.$

$z_t^1 \leftarrow a_{t-1}^2 \wedge b_{t-1}^1.$

$z_t^0 \leftarrow a_{t-1}^0.$

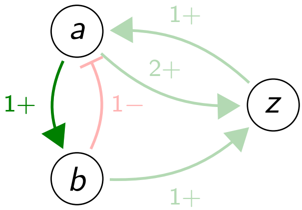
$z_t^0 \leftarrow a_{t-1}^1.$

$z_t^0 \leftarrow b_{t-1}^0.$

etc...

Discrete Model as a Logic Program

Discrete model:



$+$ Discrete parameters
or evolution functions

Logic program:

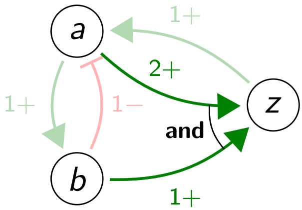
$b_t^1 \leftarrow a_{t-1}^1.$
$b_t^1 \leftarrow a_{t-1}^2.$
$b_t^0 \leftarrow a_{t-1}^0.$

$z_t^1 \leftarrow a_{t-1}^2 \wedge b_{t-1}^1.$
$z_t^0 \leftarrow a_{t-1}^0.$
$z_t^0 \leftarrow a_{t-1}^1.$
$z_t^0 \leftarrow b_{t-1}^0.$

etc...

Discrete Model as a Logic Program

Discrete model:



+ Discrete parameters
or evolution functions

Logic program:

$b_t^1 \leftarrow a_{t-1}^1.$

$b_t^1 \leftarrow a_{t-1}^2.$

$b_t^0 \leftarrow a_{t-1}^0.$

$z_t^1 \leftarrow a_{t-1}^2 \wedge b_{t-1}^1.$

$z_t^0 \leftarrow a_{t-1}^0.$

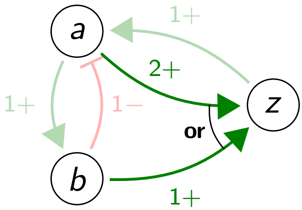
$z_t^0 \leftarrow a_{t-1}^1.$

$z_t^0 \leftarrow b_{t-1}^0.$

etc...

Discrete Model as a Logic Program

Discrete model:



+ Discrete parameters
or evolution functions

Logic program:

$b_t^1 \leftarrow a_{t-1}^1.$

$b_t^1 \leftarrow a_{t-1}^2.$

$b_t^0 \leftarrow a_{t-1}^0.$

$z_t^1 \leftarrow a_{t-1}^2.$

$z_t^1 \leftarrow b_{t-1}^1.$

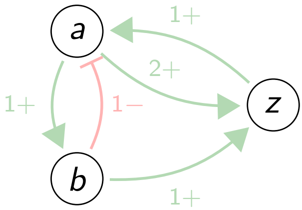
$z_t^0 \leftarrow a_{t-1}^1 \wedge b_{t-1}^0.$

$z_t^0 \leftarrow a_{t-1}^0 \wedge b_{t-1}^0.$

etc...

Discrete Model as a Logic Program

Discrete model:



+ Discrete parameters
or evolution functions

Logic program:

$b_t^1 \leftarrow a_{t-1}^1.$

$b_t^1 \leftarrow a_{t-1}^2.$

$b_t^0 \leftarrow a_{t-1}^0.$

$z_t^1 \leftarrow a_{t-1}^2.$

$z_t^1 \leftarrow b_{t-1}^1.$

$z_t^0 \leftarrow a_{t-1}^1 \wedge b_{t-1}^0.$

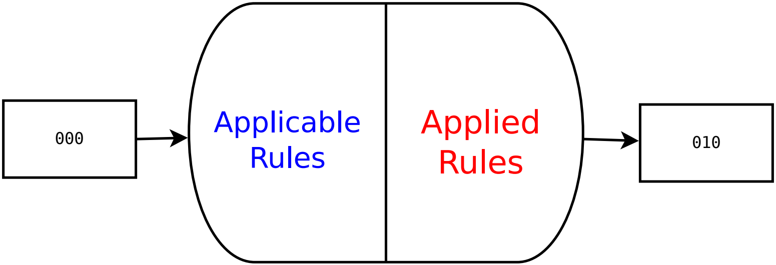
$z_t^0 \leftarrow a_{t-1}^0 \wedge b_{t-1}^0.$

etc...

Learning

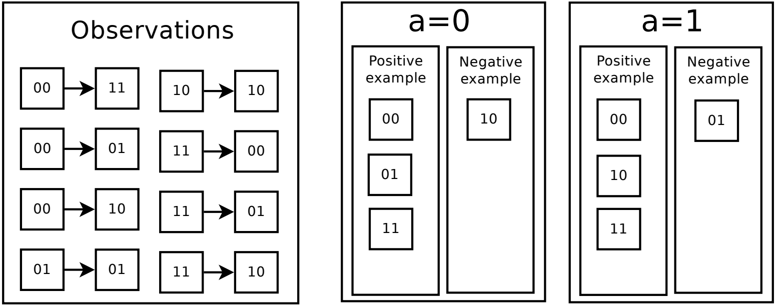
Semantics-Free Learning

Semantics = computing the next state by selecting, among applicable local rules, the ones that will be applied.



Learning Intuition: Classification Problem

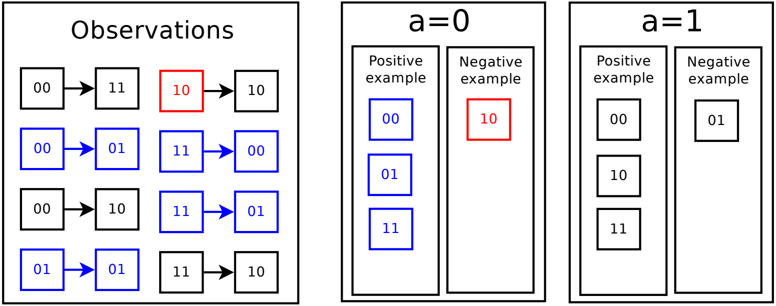
What is an applicable rule? The **conditions** so that a variable **can** take a certain value in next state.



Equivalent to a **classification problem**: for each variable value, what is a **typical state** where the variable **can** take this value in the next state ?

Learning Intuition: Classification Problem

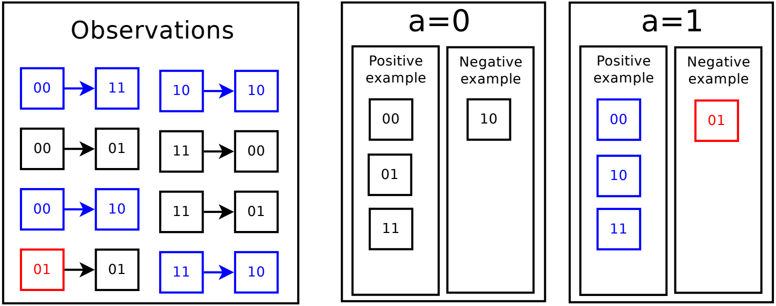
What is an applicable rule? The **conditions** so that a variable **can** take a certain value in next state.



Equivalent to a **classification problem**: for each variable value, what is a **typical state** where the variable **can** take this value in the next state ?

Learning Intuition: Classification Problem

What is an applicable rule? The **conditions** so that a variable **can** take a certain value in next state.



Equivalent to a **classification problem**: for each variable value, what is a **typical state** where the variable **can** take this value in the next state ?

GULA

GULA = General Usage LFIT Algorithm

Input: a set of transitions (feature $\rightarrow$ target)

Output: a program that respects:

- **Consistency**: the program allows no negative examples
- **Realization**: the program covers all positive examples
- **Completeness**: the program covers all the state space
- **minimality** of the rules (most general bodies)

Method: start from most general rules and **specialize** iteratively.

Least Specialization

Ensure consistency of a rule:

$$\underbrace{\mathrm{v}_0^{val_0}}_{head} \leftarrow \underbrace{\mathrm{v}_1^{val_1} \wedge \mathrm{v}_2^{val_2} \wedge \ldots \wedge \mathrm{v}_n^{val_n}}_{body}.$$

→ Used when a rule matches a negative example s: $body \subseteq s$.
→ Add **one** condition to *body* that prevents matching s.

Examples:

$$\left.\begin{array}{l} a_t^1 \leftarrow \top. \\ b_t^0 \leftarrow a_{t-1}^0. \\ ch^2 \leftarrow a_{t-1}^0 \wedge b_{t-1}^1 \wedge st^1. \end{array}\right\} \begin{array}{l} \text{all } \textbf{match } \langle a_{t-1}^0, b_{t-1}^1, st^1 \rangle \\ \rightarrow \text{how to specialize each one?} \end{array}$$

Suppose $\mathrm{dom}(a_{t-1}) = \mathrm{dom}(b_{t-1}) = \{0, 1\}$ and $\mathrm{dom}(st) = \{0, 1, 2\}$.

The Least Specialization of $a_t^1 \leftarrow \top.$ is:

$\rightarrow \quad \{ \; a_t^1 \leftarrow a_{t-1}^1. \; ; \; a_t^1 \leftarrow b_{t-1}^0. \; ; \; a_t^1 \leftarrow st^0. \; ; \; a_t^1 \leftarrow st^2. \; \}$

Least Specialization

Ensure consistency of a rule:

$$\underbrace{\mathrm{v}_0^{val_0}}_{head} \leftarrow \underbrace{\mathrm{v}_1^{val_1} \wedge \mathrm{v}_2^{val_2} \wedge \ldots \wedge \mathrm{v}_n^{val_n}}_{body}.$$

→ Used when a rule matches a negative example s: *body* $\subseteq s$.

→ Add **one** condition to *body* that prevents matching s.

Examples:

$$\left.\begin{array}{l} a_t^1 \leftarrow \top. \\ b_t^0 \leftarrow a_{t-1}^0. \\ ch^2 \leftarrow a_{t-1}^0 \wedge b_{t-1}^1 \wedge st^1. \end{array}\right\} \begin{array}{l} \text{all } \textbf{match } \langle a_{t-1}^0, b_{t-1}^1, st^1 \rangle \\ \rightarrow \text{how to specialize each one?} \end{array}$$

Suppose $\mathrm{dom}(a_{t-1}) = \mathrm{dom}(b_{t-1}) = \{0, 1\}$ and $\mathrm{dom}(st) = \{0, 1, 2\}$.

The Least Specialization of $a_t^1 \leftarrow \top.$ is:

$$\rightarrow \quad \{ \ a_t^1 \leftarrow a_{t-1}^1. \ ; \ a_t^1 \leftarrow b_{t-1}^0. \ ; \ a_t^1 \leftarrow st^0. \ ; \ a_t^1 \leftarrow st^2. \ \}$$

Least Specialization

Ensure consistency of a rule:

$$\underbrace{\mathrm{v}_0^{val_0}}_{head} \leftarrow \underbrace{\mathrm{v}_1^{val_1} \wedge \mathrm{v}_2^{val_2} \wedge \ldots \wedge \mathrm{v}_n^{val_n}}_{body}.$$

→ Used when a rule matches a negative example s: *body* $\subseteq s$.
→ Add **one** condition to *body* that prevents matching s.

Examples:

$a_t^1 \leftarrow \top.$
$b_t^0 \leftarrow a_{t-1}^0.$
$ch^2 \leftarrow a_{t-1}^0 \wedge b_{t-1}^1 \wedge st^1.$

all **match** $\langle a_{t-1}^0, b_{t-1}^1, st^1 \rangle$
→ how to specialize each one?

Suppose $\mathrm{dom}(a_{t-1}) = \mathrm{dom}(b_{t-1}) = \{0, 1\}$ and $\mathrm{dom}(st) = \{0, 1, 2\}$.

The Least Specialization of $a_t^1 \leftarrow \top.$ is:

→ $\{\ a_t^1 \leftarrow a_{t-1}^1.\ ;\ a_t^1 \leftarrow b_{t-1}^0.\ ;\ a_t^1 \leftarrow st^0.\ ;\ a_t^1 \leftarrow st^2.\ \}$

Least Specialization

Ensure consistency of a rule:

$$\underbrace{\mathrm{v}_0^{val_0}}_{head} \leftarrow \underbrace{\mathrm{v}_1^{val_1} \ \wedge \ \mathrm{v}_2^{val_2} \ \wedge \ \ldots \ \wedge \ \mathrm{v}_n^{val_n}}_{body}.$$

→ Used when a rule matches a negative example s: $body \subseteq s$.
→ Add **one** condition to *body* that prevents matching s.

Examples:

$$\left.\begin{array}{l} a_t^1 \leftarrow \top. \\ b_t^0 \leftarrow a_{t-1}^0. \\ ch^2 \leftarrow a_{t-1}^0 \wedge b_{t-1}^1 \wedge st^1. \end{array}\right\} \begin{array}{l} \text{all } \textbf{match } \langle a_{t-1}^0, b_{t-1}^1, st^1 \rangle \\ \rightarrow \text{how to specialize each one?} \end{array}$$

Suppose $\mathrm{dom}(a_{t-1}) = \mathrm{dom}(b_{t-1}) = \{0, 1\}$ and $\mathrm{dom}(st) = \{0, 1, 2\}$.

The Least Specialization of $a_t^1 \leftarrow \top.$ is:
→ $\{ \ a_t^1 \leftarrow a_{t-1}^1. \ ; \ a_t^1 \leftarrow b_{t-1}^0. \ ; \ a_t^1 \leftarrow st^0. \ ; \ a_t^1 \leftarrow st^2. \ \}$

Least Specialization

Ensure consistency of a rule:

$$\underbrace{\mathrm{v}_0^{val_0}}_{head} \leftarrow \underbrace{\mathrm{v}_1^{val_1} \wedge \mathrm{v}_2^{val_2} \wedge \ldots \wedge \mathrm{v}_n^{val_n}}_{body}.$$

→ Used when a rule matches a negative example s: $body \subseteq s$.
→ Add **one** condition to *body* that prevents matching s.

Examples:

$$\left.\begin{array}{l} a_t^1 \leftarrow \top. \\ b_t^0 \leftarrow a_{t-1}^0. \\ ch^2 \leftarrow a_{t-1}^0 \wedge b_{t-1}^1 \wedge st^1. \end{array}\right\}$$

all **match** $\langle a_{t-1}^0, b_{t-1}^1, st^1 \rangle$
→ how to specialize each one?

Suppose $\mathrm{dom}(a_{t-1}) = \mathrm{dom}(b_{t-1}) = \{0, 1\}$ and $\mathrm{dom}(st) = \{0, 1, 2\}$.

The Least Specialization of $b_t^0 \leftarrow a_{t-1}^0.$ is:

→ $\{\ b_t^0 \leftarrow a_{t-1}^0 \wedge b_{t-1}^0.\ ;\ b_t^0 \leftarrow a_{t-1}^0 \wedge st^0.\ ;\ b_t^0 \leftarrow a_{t-1}^0 \wedge st^2.\ \}$

Least Specialization

Ensure consistency of a rule:

$$\underbrace{\mathrm{v}_0^{val_0}}_{head} \leftarrow \underbrace{\mathrm{v}_1^{val_1} \wedge \mathrm{v}_2^{val_2} \wedge \dots \wedge \mathrm{v}_n^{val_n}}_{body}.$$

→ Used when a rule matches a negative example s: $body \subseteq s$.
→ Add **one** condition to *body* that prevents matching s.

Examples:

$$\left.\begin{array}{l} a_t^1 \leftarrow \top. \\ b_t^0 \leftarrow a_{t-1}^0. \\ ch^2 \leftarrow a_{t-1}^0 \wedge b_{t-1}^1 \wedge st^1. \end{array}\right\} \begin{array}{l} \text{all } \textbf{match } \langle a_{t-1}^0, b_{t-1}^1, st^1 \rangle \\ \rightarrow \text{how to specialize each one?} \end{array}$$

Suppose $\mathrm{dom}(a_{t-1}) = \mathrm{dom}(b_{t-1}) = \{0, 1\}$ and $\mathrm{dom}(st) = \{0, 1, 2\}$.

The Least Specialization of $ch^2 \leftarrow a_{t-1}^0 \wedge b_{t-1}^1, st^1.$ is:

→ $\emptyset$

GULA: General Usage LFIT Algorithm

GULA: **INPUT:** a set of transitions T.
Initialize $P = \emptyset$
For each existing target atom v^{val}

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$
- Initialize $P_{\mathrm{v}^{val}} := \{\mathrm{v}^{val} \leftarrow \top.\}$
- For each state $s \in Neg_{\mathrm{v}^{val}}$
  - Replace each rule that matches s by its least specializations
  - Remove all dominated rules, that is, that are not the most general:
    $head(R) = head(R')$ and $body(R) \subseteq body(R')$
- $P := P \cup P_{\mathrm{v}^{val}}$

OUTPUT: $P_{\mathcal{O}}(T) := P$ the optimal program of T.

Formally proved: Compatible with transitions generated in **synchronous**, **asynchronous** and **general** semantics.

Also proved: Compatible with a wider class of “learnable” semantics.

GULA: General Usage LFIT Algorithm

GULA: **INPUT:** a set of transitions T.
Initialize $P = \emptyset$
For each existing target atom v^{val}

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$
- Initialize $P_{\mathrm{v}^{val}} := \{\mathrm{v}^{val} \leftarrow \top.\}$
- For each state $s \in Neg_{\mathrm{v}^{val}}$
  - Replace each rule that matches s by its least specializations
  - Remove all dominated rules, that is, that are not the most general:
    $head(R) = head(R')$ and $body(R) \subseteq body(R')$
- $P := P \cup P_{\mathrm{v}^{val}}$

OUTPUT: $P_{\mathcal{O}}(T) := P$ the optimal program of T.

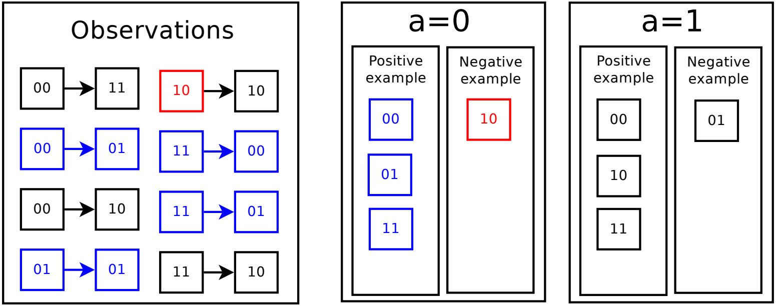
Formally proved: Compatible with transitions generated in **synchronous**, **asynchronous** and **general** semantics.

Also proved: Compatible with a wider class of “learnable” semantics.

GULA: General Usage LFIT Algorithm

GULA: **INPUT:** a set of transitions T.
Initialize $P = \emptyset$
For each existing target atom v^{val}

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$



GULA: General Usage LFIT Algorithm

GULA: **INPUT:** a set of transitions T.
Initialize $P = \emptyset$
For each existing target atom v^{val}

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$
- Initialize $P_{\mathrm{v}^{val}} := \{\mathrm{v}^{val} \leftarrow \top.\}$
- For each state $s \in Neg_{\mathrm{v}^{val}}$
  - Replace each rule that matches s by its least specializations
  - Remove all dominated rules, that is, that are not the most general:
    $head(R) = head(R')$ and $body(R) \subseteq body(R')$
- $P := P \cup P_{\mathrm{v}^{val}}$

OUTPUT: $P_{\mathcal{O}}(T) := P$ the optimal program of T.

Formally proved: Compatible with transitions generated in **synchronous**, **asynchronous** and **general** semantics.

Also proved: Compatible with a wider class of “learnable” semantics.

GULA: General Usage LFIT Algorithm

GULA: **INPUT:** a set of transitions T.
Initialize $P = \emptyset$
For each existing target atom v^{val}

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$
- Initialize $P_{\mathrm{v}^{val}} := \{\mathrm{v}^{val} \leftarrow \top.\}$
- For each state $s \in Neg_{\mathrm{v}^{val}}$
  - Replace each rule that matches s by its least specializations
  - Remove all dominated rules, that is, that are not the most general:
    $head(R) = head(R')$ and $body(R) \subseteq body(R')$
- $P := P \cup P_{\mathrm{v}^{val}}$

OUTPUT: $P_{\mathcal{O}}(T) := P$ the optimal program of T.

Formally proved: Compatible with transitions generated in **synchronous**, **asynchronous** and **general** semantics.

Also proved: Compatible with a wider class of "learnable" semantics.

GULA: General Usage LFIT Algorithm

GULA: **INPUT:** a set of transitions T.
Initialize $P = \emptyset$
For each existing target atom v^{val}

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$
- Initialize $P_{\mathrm{v}^{val}} := \{\mathrm{v}^{val} \leftarrow \top.\}$
- For each state $s \in Neg_{\mathrm{v}^{val}}$
  - Replace each rule that matches s by its least specializations
  - Remove all dominated rules, that is, that are not the most general:
    $head(R) = head(R')$ and $body(R) \subseteq body(R')$
- $P := P \cup P_{\mathrm{v}^{val}}$

OUTPUT: $P_{\mathcal{O}}(T) := P$ the optimal program of T.

Formally proved: Compatible with transitions generated in **synchronous**, **asynchronous** and **general** semantics.

Also proved: Compatible with a wider class of "learnable" semantics.

GULA: General Usage LFIT Algorithm

GULA: **INPUT:** a set of transitions T.

Initialize $P = \emptyset$

For each existing target atom v^{val}

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$
- Initialize $P_{\mathrm{v}^{val}} := \{\mathrm{v}^{val} \leftarrow \top.\}$
- For each state $s \in Neg_{\mathrm{v}^{val}}$
  - Replace each rule that matches s by its least specializations
  - Remove all dominated rules, that is, that are not the most general:
    $head(R) = head(R')$ and $body(R) \subseteq body(R')$
- $P := P \cup P_{\mathrm{v}^{val}}$

OUTPUT: $P_{\mathcal{O}}(T) := P$ the optimal program of T.

Formally proved: Compatible with transitions generated in **synchronous**, **asynchronous** and **general** semantics.

Also proved: Compatible with a wider class of "learnable" semantics.

GULA: General Usage LFIT Algorithm

GULA: INPUT: a set of transitions T.
Initialize $P = \emptyset$
For each existing target atom v^{val}

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$
- Initialize $P_{\mathrm{v}^{val}} := \{\mathrm{v}^{val} \leftarrow \top.\}$
- For each state $s \in Neg_{\mathrm{v}^{val}}$
  - Replace each rule that matches s by its least specializations
  - Remove all dominated rules, that is, that are not the most general:
    $head(R) = head(R')$ and $body(R) \subseteq body(R')$
- $P := P \cup P_{\mathrm{v}^{val}}$

OUTPUT: $P_{\mathcal{O}}(T) := P$ the optimal program of T.

Formally proved: Compatible with transitions generated in **synchronous**, **asynchronous** and **general** semantics.

Also proved: Compatible with a wider class of “learnable” semantics.

GULA: General Usage LFIT Algorithm

GULA: **INPUT:** a set of transitions T.
Initialize $P = \emptyset$
For each existing target atom v^{val}

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$
- Initialize $P_{\mathrm{v}^{val}} := \{\mathrm{v}^{val} \leftarrow \top.\}$
- For each state $s \in Neg_{\mathrm{v}^{val}}$
  - Replace each rule that matches s by its least specializations
  - Remove all dominated rules, that is, that are not the most general:
    $head(R) = head(R')$ and $body(R) \subseteq body(R')$
- $P := P \cup P_{\mathrm{v}^{val}}$

OUTPUT: $P_{\mathcal{O}}(T) := P$ the optimal program of T.

Formally proved: Compatible with transitions generated in **synchronous**, **asynchronous** and **general** semantics.

Also proved: Compatible with a wider class of "learnable" semantics.

GULA: General Usage LFIT Algorithm

GULA: **INPUT:** a set of transitions T.
Initialize $P = \emptyset$
For each existing target atom v^{val}

- Extract all states from which no transition to v^{val} exist:
  $Neg_{\mathrm{v}^{val}} := \{s \mid \nexists (s, s') \in T, \mathrm{v}^{val} \in s'\}$
- Initialize $P_{\mathrm{v}^{val}} := \{\mathrm{v}^{val} \leftarrow \top.\}$
- For each state $s \in Neg_{\mathrm{v}^{val}}$
  - Replace each rule that matches s by its least specializations
  - Remove all dominated rules, that is, that are not the most general:
    $head(R) = head(R')$ and $body(R) \subseteq body(R')$
- $P := P \cup P_{\mathrm{v}^{val}}$

OUTPUT: $P_{\mathcal{O}}(T) := P$ the optimal program of T.

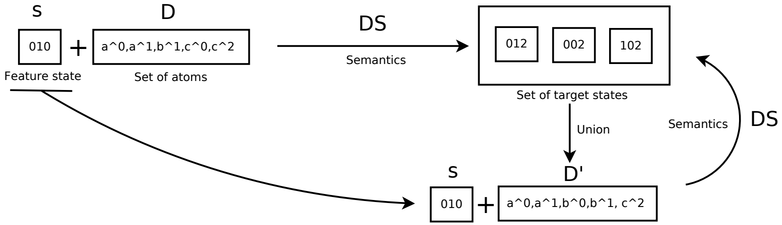
Formally proved: Compatible with transitions generated in **synchronous**, **asynchronous** and **general** semantics.

Also proved: Compatible with a wider class of “learnable” semantics.

Learnable Semantics: Pseudo-Idempotent

→ Consider a function *DS* that maps a feature state and a set of target atoms to a set of target states

→ Such that given the same state and the union of its output, it produces the same result (pseudo-indempotent)



→ A program gives possible target values (D)

→ A semantics gives which combinations are possible ($DS(s, D)$)

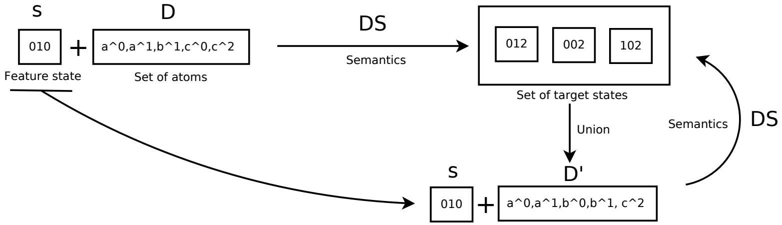
→ If the semantics produces the same states given those local values, then **GULA** learns a programs equivalent to the original one under this semantics:

$$DS(s, D) = DS(s, D') \implies DS(P) = DS(GULA(DS(P)))$$

Learnable Semantics: Pseudo-Idempotent

→ Consider a function DS that maps a feature state and a set of target atoms to a set of target states

→ Such that given the same state and the union of its output, it produces the same result (pseudo-indempotent)



→ A program gives possible target values (D)

→ A semantics gives which combinations are possible ($DS(s, D)$)

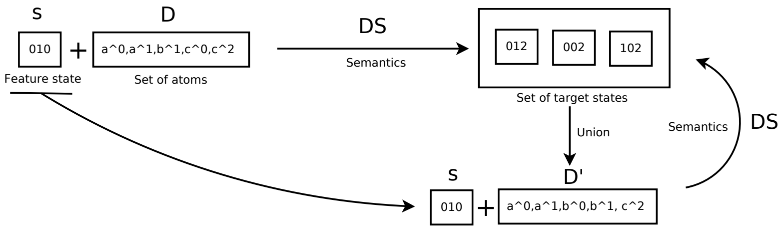
→ If the semantics produces the same states given those local values, then **GULA** learns a programs equivalent to the original one under this semantics:

$$DS(s, D) = DS(s, D') \implies DS(P) = DS(GULA(DS(P)))$$

Learnable Semantics: Pseudo-Idempotent

→ Consider a function *DS* that maps a feature state and a set of target atoms to a set of target states

→ Such that given the same state and the union of its output, it produces the same result (pseudo-indempotent)



→ A program gives possible target values (D)

→ A semantics gives which combinations are possible ($DS(s, D)$)

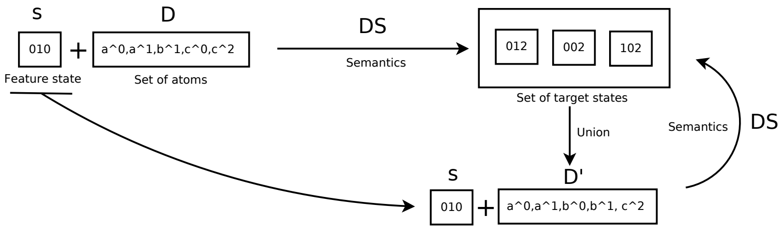
→ If the semantics produces the same states given those local values, then **GULA** learns a programs equivalent to the original one under this semantics:

$$DS(s, D) = DS(s, D') \implies DS(P) = DS(GULA(DS(P)))$$

Learnable Semantics: Pseudo-Idempotent

→ Consider a function *DS* that maps a feature state and a set of target atoms to a set of target states

→ Such that given the same state and the union of its output, it produces the same result (pseudo-indempotent)



→ A program gives possible target values (D)

→ A semantics gives which combinations are possible ($DS(s, D)$)

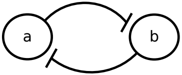
→ If the semantics produces the same states given those local values, then **GULA** learns a programs equivalent to the original one under this semantics:

$$DS(s, D) = DS(s, D') \implies DS(P) = DS(GULA(DS(P)))$$

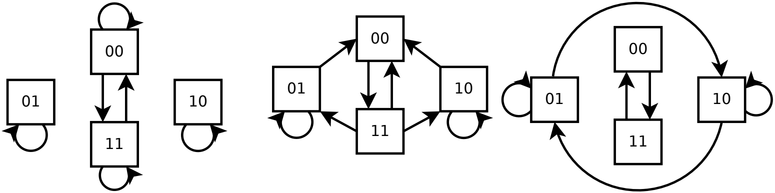
Learning Semantics

What if we don't know the semantics?

Three examples of arbitrary semantics:



f(a) := not b.
f(b) := not a.

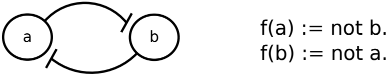


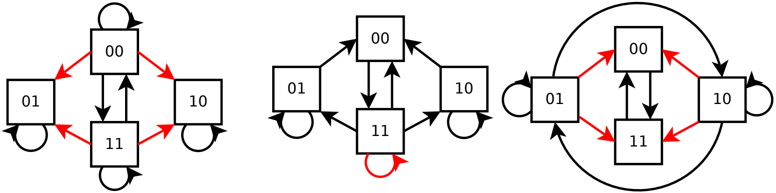
All or nothing change   Degradation   Inverse all values

How can we learn a program able to reproduce such behavior?

What is impossible?

If we use the program learned by **GULA** with the synchronous semantics, we observe spurious transitions, which were not in the observations:





All or nothing change

Degradation

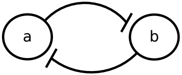
Inverse all values

How to prevent these impossible transitions?

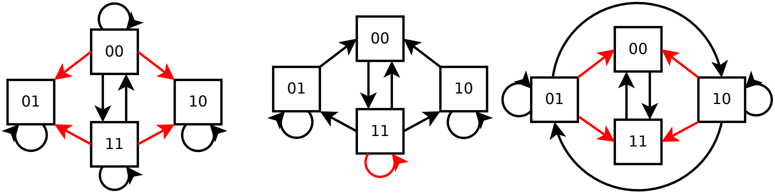
We need "impossibility rules": **constraints**!

What is impossible?

If we use the program learned by **GULA** with the synchronous semantics, we observe spurious transitions, which were not in the observations:



f(a) := not b.
f(b) := not a.

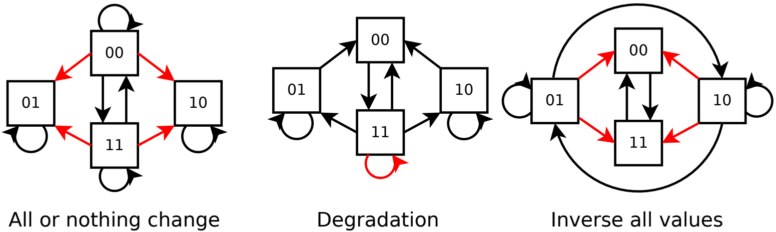


All or nothing change   Degradation   Inverse all values

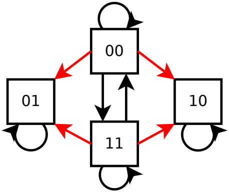
How to prevent these impossible transitions?

We need "impossibility rules": **constraints**!

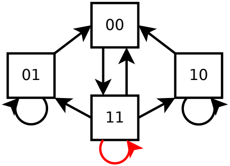
Classification Modeling of Impossibility



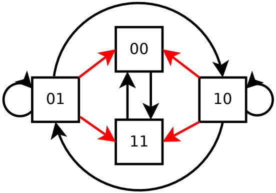
Classification Modeling of Impossibility



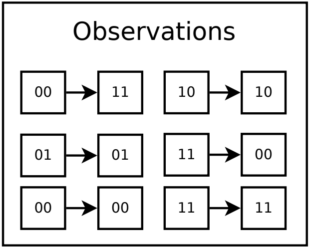
All or nothing change

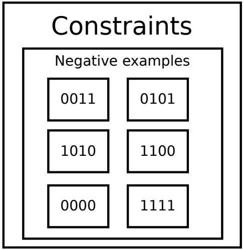


Degradation



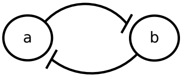
Inverse all values



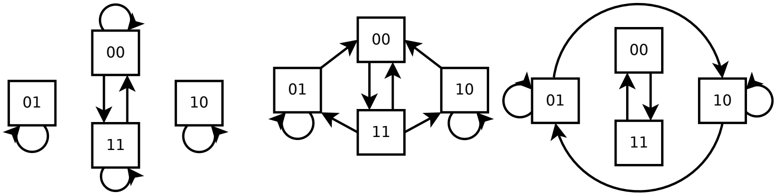


Learning Any Semantics Dynamics

- **INPUT:** T, a set of transitions produced using **any semantics**.



f(a) := not b.
f(b) := not a.



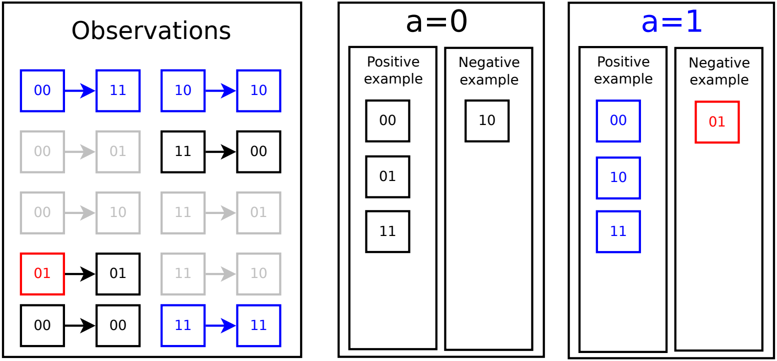
All or nothing change

Degradation

Inverse all values

Learning Any Semantics Dynamics

- **INPUT:** T, a set of transitions produced using **any semantics**.
- From T, learn a program P using GULA: gives local influences and possible values of each variables (including spurious transitions)



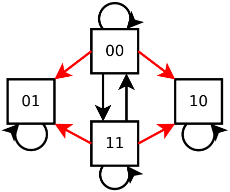
Learning Any Semantics Dynamics

- **INPUT:** T, a set of transitions produced using **any semantics**.
- From T, learn a program P using GULA: gives local influences and possible values of each variables (including spurious transitions)

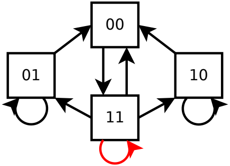
a := not b
a(0,T) :- b(1,T-1).
a(1,T) :- b(0,T-1).
b := not a
b(0,T) :- a(1,T-1).
b(1,T) :- a(0,T-1).
Conservation rules
a(0,T) :- a(0,T-1).
a(1,T) :- a(1,T-1).
b(0,T) :- b(0,T-1).
b(1,T) :- b(1,T-1).

Learning Any Semantics Dynamics

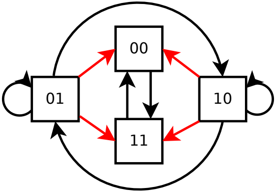
- **INPUT:** T, a set of transitions produced using **any semantics**.
- From T, learn a program P using GULA: gives local influences and possible values of each variables (including spurious transitions)
- Encode T into negative examples of constraint matching



All or nothing change



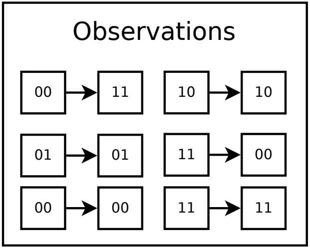
Degradation

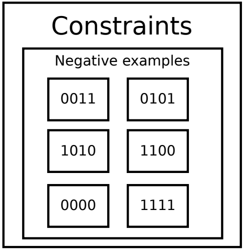


Inverse all values

Learning Any Semantics Dynamics

- **INPUT:** T, a set of transitions produced using **any semantics**.
- From T, learn a program P using GULA: gives local influences and possible values of each variables (including spurious transitions)
- Encode T into negative examples of constraint matching





Learning Any Semantics Dynamics

- **INPUT:** T, a set of transitions produced using **any semantics**.
- From T, learn a program P using GULA: gives local influences and possible values of each variables (including spurious transitions)
- Encode T into negative examples of constraint matching
- Learn a program P' using GULA from this encoding: P' contains all minimal constraints covering impossible transitions

Constraints

```
:- a(0,T), b(1,T), b(0,T-1).
:- a(1,T), b(0,T), a(0,T-1).
:- a(1,T), b(0,T), b(1,T-1).
:- a(0,T), b(1,T), a(1,T-1).
...
```

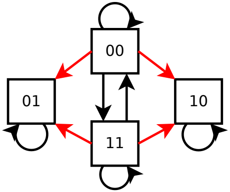
Learning Any Semantics Dynamics

- **INPUT:** T, a set of transitions produced using **any semantics**.
- From T, learn a program P using GULA: gives local influences and possible values of each variables (including spurious transitions)
- Encode T into negative examples of constraint matching
- Learn a program P' using GULA from this encoding: P' contains all minimal constraints covering impossible transitions
- Discard in P' inapplicable constraints according to P

Learning Any Semantics Dynamics

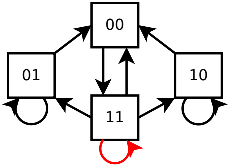
- **INPUT:** T, a set of transitions produced using **any semantics**.
- From T, learn a program P using GULA: gives local influences and possible values of each variables (including spurious transitions)
- Encode T into negative examples of constraint matching
- Learn a program P' using GULA from this encoding: P' contains all minimal constraints covering impossible transitions
- Discard in P' inapplicable constraints according to P
- **OUTPUT:** $P \cup P'$ which exactly reproduces T, under the **constrained synchronous semantics**

Examples of learned programs



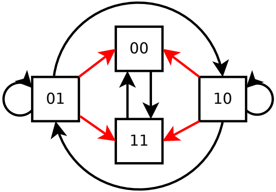
All or nothing change

a := not b
a(0,T) :- b(1,T-1).
a(1,T) :- b(0,T-1).
b := not a
b(0,T) :- a(1,T-1).
b(1,T) :- a(0,T-1).
Conservation rules
a(0,T) :- a(0,T-1).
a(1,T) :- a(1,T-1).
b(0,T) :- b(0,T-1).
b(1,T) :- b(1,T-1).
Constraints
:- a(0,T), b(1,T), b(0,T-1).
:- a(1,T), b(0,T), a(0,T-1).
:- a(1,T), b(0,T), b(1,T-1).
:- a(0,T), b(1,T), a(1,T-1).



Degradation

a := not b
a(0,T) :- b(1,T-1).
a(1,T) :- b(0,T-1).
b := not a
b(0,T) :- a(1,T-1).
b(1,T) :- a(0,T-1).
Conservation rules
a(1,T) :- a(1,T-1).
b(1,T) :- b(1,T-1).
Degradation
a(0,T) :- a(1,T-1).
b(0,T) :- b(1,T-1).
Constraints
:- a(1,T), b(1,T), a(1,T-1).



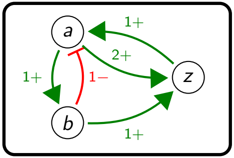
Inverse all values

a := not b
a(0,T) :- b(1,T-1).
a(1,T) :- b(0,T-1).
b := not a
b(0,T) :- a(1,T-1).
b(1,T) :- a(0,T-1).
Inverse value
a(0,T) :- a(1,T-1).
a(1,T) :- a(0,T-1).
b(0,T) :- b(1,T-1).
b(1,T) :- b(0,T-1).
Constraints
:- a(1,T), b(1,T), a(1,T-1).
:- a(0,T), b(0,T), a(0,T-1).
:- a(1,T), b(1,T), b(1,T-1).
:- a(0,T), b(0,T), b(0,T-1).

Learning Time Series

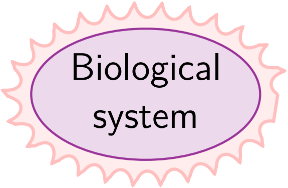
Potential Usage

Black box $\xrightarrow{\text{Semantics}}$ State graph

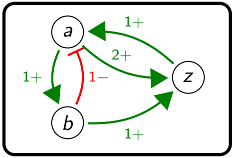


$\xrightarrow{\text{Semantics}}$ State graph

Potential Usage

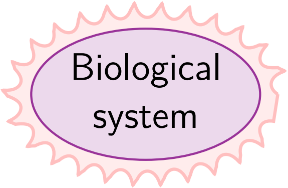


Semantics → State graph

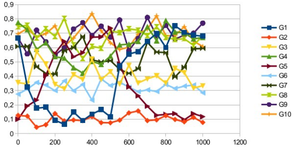


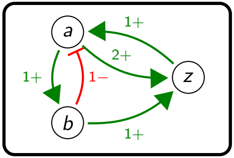
Semantics → State graph

Potential Usage



Behavior →

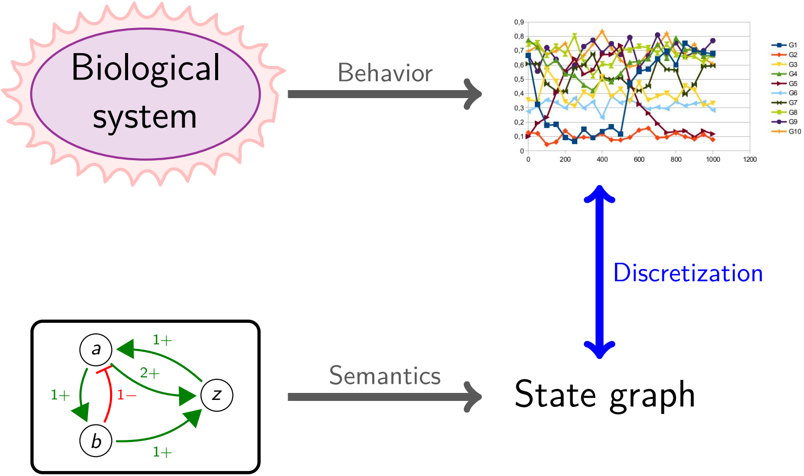




Semantics →

State graph

Potential Usage



Scalability of GULA

Run time of **GULA** for 9 to 18 nodes Boolean networks for the three semantics: run time in seconds for 25%/50%/75%/100% of the transitions as input, and total number of transitions.

Benchmark	size	synchronous	asynchronous	general
arellano_rootstem	9	2s/1.8s/0.9s/0.3s/512	2.4s/1.4s/1.1s/0.2s/1,940	1.1s/0.5s/0.3s/0.3s/11*K*
davidich_yeast	10	16s/10s/4s/0.6s/1,024	12s/6s/4s/0.5s/4,364	3s/1.5s/1s/0.9s/39*K*
faure_cellcycle	10	15s/10s/4s/0.8s/1,024	12s/5.6s/4.7s/0.6s/4,273	4s/1.2s/0.9s/0.9s/31*K*
fission_yeast	10	16s/10s/4.8s/0.8s/1,024	12s/5.8s/4.6s/0.4s/4,157	3.6s/1.2s/1s/0.8s/34*K*
mammalian	10	14.8s/11s/4.8s/0.8s/1,024	12s/5.7s/3.4s/0.6s/4,273	3.4s/1.4s/1s/0.9s/31*K*
budding_yeast	12	564s/194s/61s/3.7s/4,096	216s/107s/85s/2.6s/20*K*	51s/14s/5.9s/4.1s/260*K*
n12c5	12	468s/200s/64s/2.8s/4,096	213s/103s/144s/1.3s/30*K*	4.7s/6s/8.6s/11s/1,122*K*
tournier_apoptosis	12	369s/164s/54s/2.7s/4,096	199s/98s/94s/2s/22*K*	26s/6.7s/4.6s/4.6s/358*K*
dinwoodie_stomatal	13	-/748s/221s/6.1s/8,192	-/548s/628s/4s/53*K*	70s/18s/15s/18s/1.5*M*
multivalued	13	-/-/406s/6s/8,192	-/565s/765s/4.9s/49*K*	61s/18s/13s/13s/1*M*
saadatpour_guardcell	13	-/757s/219s/6s/8,192	-/575s/638s/4.2s/53*K*	68s/17s/15s/18s/1.5*M*
arabidopsis	15	-/-/-/53s/32*K*	-/-/-/50s/213*K*	-/352s/123s/103s/7*M*
dinwoodie_life	15	-/-/-/37s/32*K*	-/-/-/30s/245*K*	-/352s/240s/256s/20*M*
randomnet_n15k3	15	-/-/-/51s/32*K*	-/-/-/31s/262*K*	731s/219s/226s/280s/22*M*
irons_yeast	18	-/-/-/653s/262*K*	-/-/-/324s/2*M*	memory out

Exponential w.r.t variables/values but faster if more observations.
Runtime is not a problem with **PRIDE**, a polynomial approximation.

Polynomial Approximation: PRIDE

PRIDE = Polynomial Relational Inference of Discrete Events

Input: a set of transitions (feature → target)

Output: a program that respects:

- **Consistency**: The program allows no negative examples
- **Realization**: The program covers all positive examples
- ~~**Completeness**: The program covers all the state space~~
- **Minimality** of the rules (most general bodies)

Method:
→ Keep only one specialization according to a non-matched positive example.
→ Use greedy search to minimize rules.

Learning Semantics is exponential

Run time of **Synchronizer** for 6 to 10 nodes Boolean networks for the three semantics: run time in seconds for 25%/50%/75%/100% of the transitions as input, and total number of transitions.

Benchmark	size	synchronous	asynchronous	general
n6s1c2	6	0.2s/0.3s/0.2s/0.1s/64	2.5s/4.4s/3.6s/1s/230	9s/6s/2.9s/0.5s/1,039
n7s3	7	1.6s/3.1s/2.5s/0.3s/128	32s/35s/26s/5s/451	139s/68s/21s/6s/2,243
randomnet_n7k3	7	5.9s/16s/19s/6.6s/128	25s/47s/32s/5.4s/394	133s/93s/45s/9.9s/1,580
xiao_wnt5a	7	0.96s/1.4s/1s/0.2s/128	11s/21s/12s/3s/324	25s/14s/7s/1.1s/972
arellano_rootstem	9	86s/83s/40s/2.6s/512	-/-/-/145s/1,940	-/-/-/41s/11,472
davidich_yeast	10	-/796s/363s/28s/1,024	-/-/-/622s/4,364	-/-/-/-/38,720
faure_cellcycle	10	-/-/558s/31s/1,024	-/-/-/865s/4,273	-/-/-/-/30,971
fission_yeast	10	-/-/478s/36s/1,024	-/-/-/662s/4,157	-/-/-/-/33,727
mammalian	10	-/-/598s/33s/1,024	-/-/-/841s/4,273	-/-/-/-/30,971

Prediction Power of GULA/PRIDE

Evaluate quality of rules:

→ Prediction of each variable possible value

→ Learn from partial observations (group by initial state / random)

→ Prediction from unseen states ($train \cap test = \emptyset$)

Method:

→ Use **GULA/PRIDE** to learn two programs: P and $\overline{P}$

→ P: classic program that say when a target atom is possible

→ $\overline{P}$: a kind of anti-program that say when a target atom is not possible

→ Rules are weighted by the number of observations they match

→ Probabilities can be obtain from the most matching rule/anti-rule

Predicting probabilities of a_t^0 from $\langle a_{t-1}^1, b_{t-1}^1, c_{t-1}^1, st^1 \rangle$

P:

$(105) : a_t^0 \leftarrow b_{t-1}^0.$

$(42) : a_t^0 \leftarrow b_{t-1}^1 \wedge c_{t-1}^1.$

$(12) : a_t^0 \leftarrow c_{t-1}^1 \wedge st^1.$

$\overline{P}$:

$(81) : a_t^0 \leftarrow b_{t-1}^0.$

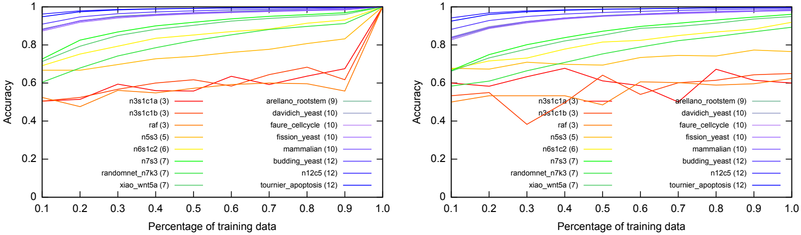
$(61) : a_t^0 \leftarrow a_{t-1}^1 \wedge c_{t-1}^0.$

$(30) : a_t^0 \leftarrow a_{t-1}^1 \wedge st^1.$

Prediction: $0.5 + 0.5 \times \frac{42-30}{42+30} = 0.58$

Accuracy: mean absolute error VS Ground truth: $0 : 0.58, 1 : 0.42$

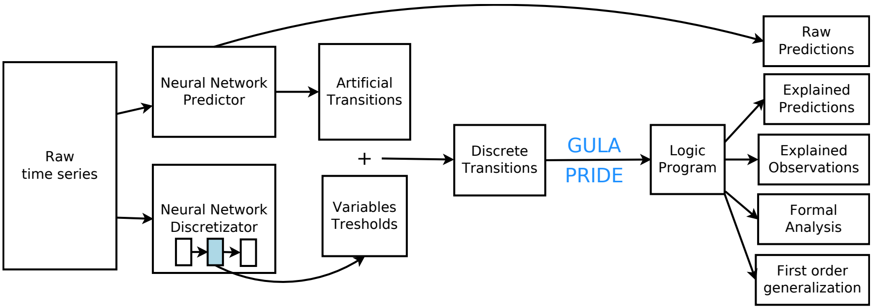
Prediction power



Partial initial states | Partial transitions

Figure: Accuracy of the models learned by **GULA** when predicting possible target variable values from unseen states: (left) experiment 1, with a complete set of input transitions from a partial number of initial states; and (right) experiment 2, with a potentially incomplete set of input transitions from an incomplete set of initial states.

Outlook: GULA/PRIDE Workflow



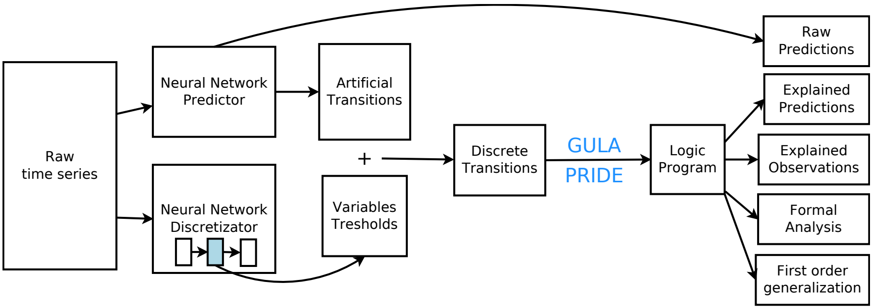
→ Pre-process: Use statistical ML for data augmentation/noise tolerance

→ Pre-process: Automatic discretization using hand-made NN layer

→ Post-process: Weight rules for predictions

→ Post-process: First order generalization to simplify explanations

Outlook: GULA/PRIDE Workflow



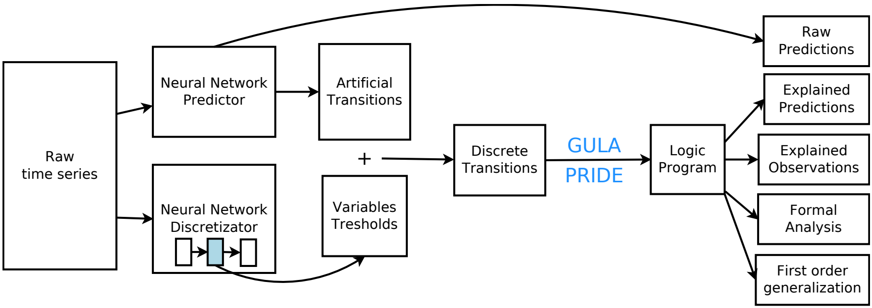
→ Pre-process: Use statistical ML for data augmentation/noise tolerance

→ Pre-process: Automatic discretization using hand-made NN layer

→ Post-process: Weight rules for predictions

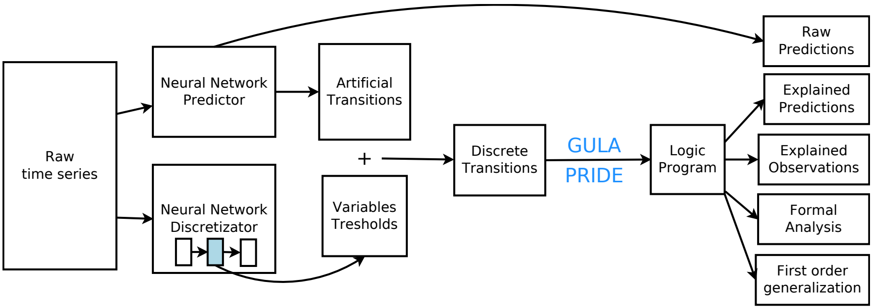
→ Post-process: First order generalization to simplify explanations

Outlook: GULA/PRIDE Workflow



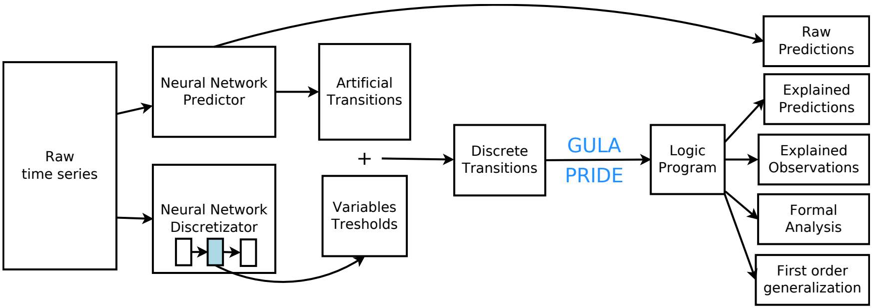
- → Pre-process: Use statistical ML for data augmentation/noise tolerance
- → Pre-process: Automatic discretization using hand-made NN layer
- → Post-process: Weight rules for predictions
- → Post-process: First order generalization to simplify explanations

Outlook: GULA/PRIDE Workflow



→ Pre-process: Use statistical ML for data augmentation/noise tolerance
→ Pre-process: Automatic discretization using hand-made NN layer
→ Post-process: Weight rules for predictions
→ Post-process: First order generalization to simplify explanations

Outlook: GULA/PRIDE Workflow



→ Pre-process: Use statistical ML for data augmentation/noise tolerance
→ Pre-process: Automatic discretization using hand-made NN layer
→ Post-process: Weight rules for predictions
→ Post-process: First order generalization to simplify explanations

Conclusion

Logic rules $\Leftrightarrow$ networks interactions $\Leftrightarrow$ automata transitions

Learning of the structure of a model
1-step learning algorithm by successive refinements

Independent of the semantics
Proved for pseudo-idempotent semantics
$\rightarrow$ Includes synchronous, asynchronous, general semantics

Outlooks

- Automatic learning of time series data (noise, discretization, ...)
- Learning probabilistic models
- Improve explainability (first order, post-processing)
- Optimizations (parallelization, approximations)

Thank you

All algorithms are open-source at:
`https://github.com/Tony-sama/pylfit`

Our questions:

- How to automatically and meaningfully discretize?
- Do you know a metrics to evaluate prediction on sets of states?
- Do you have datasets to apply GULA/PRIDE on?

Your questions?

References

- Stuart A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, volume 22, n. 3, pages 437–467, 1969.

- René Thomas. Boolean formalization of genetic control circuits. *Journal of Theoretical Biology*, volume 42, n. 3, pages 563–85, 1973.

- Katsumi Inoue, Tony Ribeiro, and Chiaki Sakama. Learning from interpretation transition. *Machine Learning Journal*, volume 94, issue 1, pages 51–79, 2014.

- Tony Ribeiro, Morgan Magnin, Katsumi Inoue, Chiaki Sakama. Learning delayed influences of biological systems. *Frontiers in Bioengineering and Biotechnology*, volume 2, issue 81, 2015.

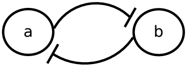
- David Martinez, Tony Ribeiro, Katsumi Inoue, Guillem Alenya, Carme Torras. Learning probabilistic action models from interpretation transitions. *The 31st International Conference on Logic Programming (ICLP)*, Cork, Ireland, 2015.

- Tony Ribeiro, Sophie Tourret, Maxime Folschette, Morgan Magnin, Domenico Borzacchiello, Francisco Chinesta, Olivier Roux, Katsumi Inoue. Inductive Learning from State Transitions over Continuous Domains. *The 27th International Conference on Inductive Logic Programming (ILP)*, Orléans, France, 2017.

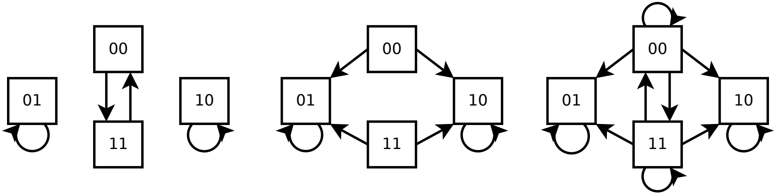
- Tony Ribeiro, Maxime Folschette, Morgan Magnin, Olivier Roux, Katsumi Inoue. Learning Dynamics with Synchronous, Asynchronous and General Semantics. *The 27th International Conference on Inductive Logic Programming (ILP)*, Ferrara, Italy, 2018.

Characterization of Classical Semantics

The three semantics can be detected by checking the following properties.



f(a) := not b.
f(b) := not a.

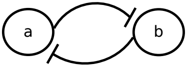


Synchronous Asynchronous General

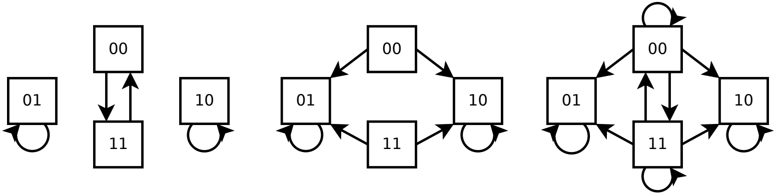
Synchronous:
$\forall (s, s_1), (s, s_2) \in T, \forall s_3 \in \mathcal{S}^{\mathcal{T}}, s_3 \subseteq s_1 \cup s_2 \implies (s, s_3) \in T.$

Characterization of Classical Semantics

The three semantics can be detected by checking the following properties.



f(a) := not b.
f(b) := not a.

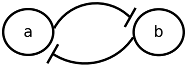


Synchronous Asynchronous General

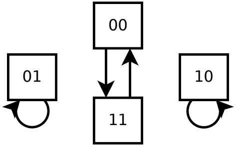
Asynchronous: $\forall (s, s') \in T, \mathrm{sp}_{\overline{\mathcal{F}} \to \overline{\mathcal{T}}}(s) \not\subseteq s', \big((s, s'') \in T, \mathrm{sp}_{\overline{\mathcal{F}} \to \overline{\mathcal{T}}}(s) \subseteq s'' \implies (s, s') \notin T\big) \wedge \big((s, s') \in T \implies |\mathrm{sp}_{\overline{\mathcal{F}} \to \overline{\mathcal{T}}}(s) \setminus s'| = 1\big)$.

Characterization of Classical Semantics

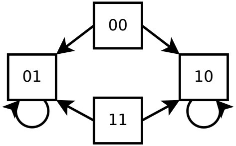
The three semantics can be detected by checking the following properties.



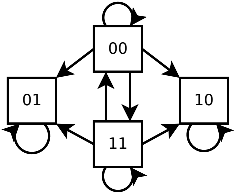
f(a) := not b.
f(b) := not a.



Synchronous



Asynchronous



General

General:
$$\forall (s, s_1), (s, s_2) \in T, \forall s_3 \in \mathcal{S}^{\mathcal{T}}, s_3 \subseteq \mathrm{sp}_{\overline{\mathcal{F}} \to \overline{\mathcal{T}}}(s) \cup s_1 \cup s_2 \implies (s, s_3) \in T.$$

Pseudo-Idempotent Semantics

Definitions:

- $\mathcal{A}_\mathcal{T}$ = all feature atoms
- $\mathcal{S}^\mathcal{F}$ = all states on feature atoms
- $\mathcal{S}^\mathcal{T}$ = all states with target atoms
- $\mathrm{Ccl}(s, P)$ = set of heads of rules in P that match s
- $P_\mathcal{O}(P)$ = optimal program (learned by GULA)

Theorem 2 (Pseudo-idempotent Semantics and Optimal $\mathcal{DM}\mathrm{VLP}$)
Let DS be a dynamical semantics.
For all P a $\mathcal{DM}\mathrm{VLP}$, if:
$\exists \mathrm{pick} \in (\mathcal{S}^\mathcal{F} \times \wp(\mathcal{A}_\mathcal{T}) \to \wp(\mathcal{S}^\mathcal{T}) \setminus \{\emptyset\})$ so that

1. $\forall s \in \mathcal{S}^\mathcal{F}, \forall D \subseteq \mathcal{A}_\mathcal{T}, \mathrm{pick}(s, \bigcup_{s' \in \mathrm{pick}(s,D)} s') = \mathrm{pick}(s, D)$ and
2. $\forall s \in \mathcal{S}^\mathcal{F}, \big(DS(P)\big)(s) = \mathrm{pick}(s, \mathrm{Ccl}(s, P))$,

then: for all P a $\mathcal{DM}\mathrm{VLP}$, $DS(P_\mathcal{O}(DS(P))) = DS(P)$.